Campus de l'Université de Montréal
C.P. 6128, succursale Centre-ville
Montréal QC H3C 3J7 Canada

Telephone : (514) 343-7575
Telecopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

CRT-2001-37

Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

A TABU SEARCH WITH SLOPE SCALING FOR THE MULTICOMMODITY CAPACITATED LOCATION PROBLEM WITH BALANCING REQUIREMENTS

by

Bernard Gendron
Jean-Yves Potvin
Patrick Soriano

October 2001

PUBLICATION
CRT-2001-37

A TABU SEARCH WITH SLOPE SCALING FOR THE MULTICOMMODITY CAPACITATED LOCATION PROBLEM WITH BALANCING REQUIREMENTS*

by

Bernard Gendron[1,2]
Jean-Yves Potvin[1,2]
Patrick Soriano[1,3]

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

1 Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7
2 Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7
3 Service de l'enseignement des méthodes quantitatives de gestion, École des Hautes Études Commerciales, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2001

ABSTRACT

In this paper, a tabu search heuristic is combined with slope scaling to solve a discrete depot location problem, known as the multicommodity location problem with balancing requirements. Although the uncapacitated version of this problem has already been addressed in the literature, this is not the case for the more challenging capacitated version, where each depot has a fixed and finite capacity. The slope scaling approach is used during the initialization phase to provide the tabu search with good starting solutions. Numerical results are reported on various types of large-scale randomly generated instances. The quality of the heuristic is assessed by comparing the solutions obtained with those of a commercial mixed-integer programming code.

Key words : multicommodity capacitated location with balancing requirements, tabu search, slope scaling

RÉSUMÉ

Nous présentons une méthode de recherche avec tabous, combinée à une approche d'ajustement de pente, pour résoudre un problème discret de localisation de dépôts, le problème de localisation multi-produits avec exigences d'équilibrage. Bien que la version du problème où il n'y a pas de capacité ait déjà été traitée, le problème plus difficile où il y une capacité sur le volume que peut entreproser chaque dépôt n'a fait l'objet d'aucun travail à ce jour. L'approche d'ajustement de pente est utilisée dans une phase initiale afin de fournir de bonnes solutions de départ pour la recherche avec tabous. Nous présentons des résultats numériques pour des problèmes de grande taille générés aléatoirement et ayant des caractéristiques diverses. Les performances de la méthode sont mesurées par rapport à celles obtenues en résolvant ces problèmes à l'aide d'un logiciel commercial de programmation en nombres entiers.

Mots clés : problème de localisation multi-produits avec capacité et exigences d'équilibrage, recherche avec tabous, ajustement de pente

103-03079/

1 Introduction

This paper describes a heuristic approach for solving a discrete location problem known as the multicommodity location problem with balancing requirements [1]. This problem comes from an industrial application related to the management of a heterogeneous fleet of containers for an international maritime shipping company. Once a ship arrives at the port, the company has to deliver the loaded containers, which may come in several types and sizes, to designated in-land destinations. Following their unloading by the importing customer, empty containers are moved to a depot. Later on, they might be delivered to customers requesting containers for subsequent shipping of their own products. Due to regional unbalances in empty container availabilities and needs throughout the network, balancing movements of empty containers among the depots are required. The problem is thus to locate the depots that will collect the supply of empty containers to satisfy the demand for empty containers, while minimizing the total operating costs. These include the cost of opening and operating the depots, and the cost generated by customer-depot and interdepot movements.

The presence of balancing movements differentiates this problem from classical discrete location applications. Different exact and heuristic approaches have been proposed for the uncapacitated version of the problem [2, 3, 4, 5, 6, 7]. However, a more challenging capacitated version, where each depot has a fixed and finite capacity, has not been addressed yet to the best of our knowledge (see [13] for a review on the related capacitated plant location problem). Although the multicommodity capacitated location problem with balancing requirements (MCLB) can be formulated as a mixed-integer programming (MIP) model, the experimental results reported in Section 5 show that large-scale instances (with up to 200 depot locations, 500 customers and 20 commodities) cannot be solved optimally by state-of-the-art MIP solvers within reasonable time and memory limits. In this paper, we combine tabu search with slope scaling to find good solutions to large-scale instances of the MCLB.

Tabu search is a well-known metaheuristic designed to explore large combinatorial solution spaces. It exploits both short and long-term memories about the search process to escape from local optima and identify new promising search paths (in the remainder, we assume a knowledge of the basic principles of tabu search; see [8] for details). The main contribution of our tabu search is a *neighborhood reduction* procedure based on effective approximations of the impact of each move on the current solution. These approximations are then used to sample a subset of the entire neighborhood for exact evaluation by a linear programming (LP) solver. Although this approach is very effective in identifying the best neighbors of the current solution (see Section 5 for computational evidence), its convergence is rather slow if it is initiated with a relatively poor solution. This is not surprising given that each exact neighbor evaluation requires the solution of a multicommodity minimum cost network flow problem (MMCF), which is computationally heavy even for state-of-the-art LP solvers. Therefore, a slope scaling method is used to provide a good initial solution to the tabu search. This method has recently been proposed to address non-convex piecewise linear cost network flow problems [10, 11, 12]. The proposed adaptation consists of an iterative procedure, where an MMCF with modified linear costs is solved at each iteration. For each arc-commodity pair, the linear transportation cost associated with that pair is modified by adding, for each depot incident to the arc, the ratio of its fixed cost on the total volume in-transit through the depot at the previous iteration. The resulting iterative scheme quickly converges to a good initial solution, which is then improved by tabu search.

The organization of the paper is the following. In Section 2, a mathematical formulation of the MCLB is proposed. This formulation is exploited to isolate the MMCF, which is used at each iteration of the slope scaling procedure, as well as for the evaluation of neighboring solutions in the tabu search. Descriptions of the tabu search and slope scaling procedures are found in Sections 3 and 4, respectively. In Section 5, computational experiments are reported and analyzed for various types of large-scale randomly generated test problems, as obtained with the generator described in the Appendix. The quality of the heuristic is assessed by comparing the solutions obtained with those found by a commercial state of the art MIP code. Finally, concluding remarks are made in Section 6.

2 Problem Formulation

To formulate the problem, we consider a directed network $G = (N, A)$, where N is the set of nodes and A is the set of arcs. There are several commodities (types of containers) moving through the network which are represented by set K. The set of nodes can be partitioned into the set of customer nodes C and the set of depots D. For each depot $j \in D$, we define its supply and demand customers as $C_j^s = \{i \in C : (i,j) \in A\}$ and $C_j^d = \{i \in C : (j,i) \in A\}$, respectively. We also assume that there is at least one supply or demand customer adjacent to each depot, that is, $C_j^s \cup C_j^d \neq \emptyset, \forall j \in D$. The sets of all supply and demand customers thus correspond to $C^s = \bigcup_{j \in D} C_j^s$ and $C^d = \bigcup_{j \in D} C_j^d$, respectively. For each node $i \in N$ (depot or customer), we also define the set of depots adjacent to this node in both directions $D_i^- = \{j \in D : (j,i) \in A\}$ and $D_i^+ = \{j \in D : (i,j) \in A\}$.

Since there are no arcs between pairs of customers, the set of arcs can be partitioned into three subsets:

- customer-to-depot arcs $A_{CD} = \{(i,j) \in A : i \in C,\ j \in D\}$;
- depot-to-customer arcs $A_{DC} = \{(j,i) \in A : j \in D,\ i \in C\}$;
- depot-to-depot arcs $A_{DD} = \{(l,j) \in A : l \in D,\ j \in D\}$.

The problem consists of minimizing the costs incurred by moving flows of commodities through the network to satisfy the supplies at origins and the demands at destinations. For each customer $i \in C^s$, the supply of commodity k is noted s_i^k, while for each customer $i \in C^d$, the demand for commodity k is noted d_i^k. All supplies and demands are assumed to be non-negative and deterministic. A non-negative transportation cost c_{ij}^k is incurred for each unit of flow of commodity k moving on arc (i,j). In addition, for each depot $j \in D$, a non-negative fixed cost f_j is incurred if the depot is opened. The problem is further complicated by the presence of a fixed capacity q_j on the volume of all commodities which can transit through depot $j \in D$, where the volume of one unit of commodity k is noted v_k.

Let x_{ij}^k represent the flow of commodity k moving on arc (i,j), and y_j be the binary location variable with value 1 if depot j is opened, and value 0 otherwise. The problem is then formulated as:

$$Z = \min \sum_{j \in D} f_j y_j + \sum_{k \in K} \left(\sum_{(i,j) \in A_{CD}} c_{ij}^k x_{ij}^k + \sum_{(j,i) \in A_{DC}} c_{ji}^k x_{ji}^k + \sum_{(l,j) \in A_{DD}} c_{lj}^k x_{lj}^k \right), \tag{1}$$

subject to

$$\sum_{j \in D_i^+} x_{ij}^k = s_i^k, \quad \forall i \in C^s, k \in K, \tag{2}$$

$$\sum_{j \in D_i^-} x_{ji}^k = d_i^k, \quad \forall i \in C^d, k \in K, \tag{3}$$

$$\sum_{i \in C_j^d} x_{ji}^k + \sum_{l \in D_j^+} x_{jl}^k - \sum_{i \in C_j^s} x_{ij}^k - \sum_{l \in D_j^-} x_{lj}^k = 0, \quad \forall j \in D, k \in K, \tag{4}$$

$$\sum_{k \in K} v^k \left(\sum_{i \in C_j^s} x_{ij}^k + \sum_{l \in D_j^-} x_{lj}^k \right) \leq q_j y_j, \quad \forall j \in D, \tag{5}$$

$$x_{ij}^k \leq s_i^k y_j, \quad \forall j \in D, i \in C_j^s, k \in K, \tag{6}$$

$$x_{ji}^k \leq d_i^k y_j, \quad \forall j \in D, i \in C_j^d, k \in K, \tag{7}$$

$$x_{ij}^k \geq 0, \quad \forall (i,j) \in A, k \in K, \tag{8}$$

$$y_j \in \{0,1\}, \quad \forall j \in D. \tag{9}$$

Constraints (2) and (3) ensure that supply and demand requirements are met. Constraints (4) and (5) are the flow conservation constraints and capacity constraints for each depot, respectively; constraints (5) also forbid customer-related movements through closed depots. The same is achieved by relations (6) and (7), which are consequently redundant. These constraints are added to the formulation because they significantly improve the quality of the lower bounds obtained through the LP relaxation. They are used when solving large-scale instances of the MCLB with a state-of-the-art MIP code (see Section 5).

Upper bounds on the optimal value of this problem can be derived by fixing both the vector of location variables y to some value $\overline{y}$ and the vector of linear transportation costs c to some value $\overline{c}$. We then obtain the following MMCF:

$$\min \sum_{k \in K} \left(\sum_{(i,j) \in A_{CD}} \overline{c}_{ij}^k x_{ij}^k + \sum_{(j,i) \in A_{DC}} \overline{c}_{ji}^k x_{ji}^k + \sum_{(l,j) \in A_{DD}} \overline{c}_{lj}^k x_{lj}^k \right), \tag{10}$$

subject to constraints (2) to (4), (8) and

$$\sum_{k \in K} v^k \left(\sum_{i \in C_j^*} x_{ij}^k + \sum_{l \in D_j^-} x_{lj}^k \right) \leq q_j \overline{y}_j, \quad \forall j \in D. \tag{11}$$

Assuming that this problem has an optimal solution, $\widetilde{x}$, an upper bound on the optimal value of the MCLB is then given by:

$$Z(\widetilde{x}, \widetilde{y}) = \sum_{j \in D} f_j \widetilde{y}_j + \sum_{k \in K} \left(\sum_{(i,j) \in A_{CD}} c_{ij}^k \widetilde{x}_{ij}^k + \sum_{(j,i) \in A_{DC}} c_{ji}^k \widetilde{x}_{ji}^k + \sum_{(l,j) \in A_{DD}} c_{lj}^k \widetilde{x}_{jl}^k \right), \tag{12}$$

where

$$\widetilde{y}_j = \begin{cases} 1, & \text{if } \left(\sum_{i \in C_j^*} \widetilde{x}_{ij}^k + \sum_{l \in D_j^-} \widetilde{x}_{lj}^k \right) > 0, \\ 0, & \text{otherwise}, \end{cases} \quad \forall j \in D. \tag{13}$$

The tabu search heuristic, which is described in the next section, solves a series of MMCFs with $\overline{c} = c$ (i.e., the transportation costs are not modified), but with y fixed to values $\overline{y}$ determined by the search procedure. The slope scaling approach presented in Section 4 considers MMCFs with all depots open (i.e., $\overline{y}_j = 1, \forall j \in D$), but modifies the linear costs $\overline{c}$ at each iteration.

3 Tabu Search

The tabu search procedure searches the space of configurations of open/closed depots, by fixing the y variables to 1 or 0 (see [4, 7], for other approaches of this type for the uncapacitated version of the problem). For each configuration, the flows on the customer-depot, depot-customer and depot-depot arcs are obtained by solving the associated MMCF. A particular configuration of open/closed depots with the corresponding flows represents a solution to the problem.

Since the customers are only linked to a subset of potential depot sites, some customers may not be connected to any open depot for a given configuration of depots, leading to infeasibility. Likewise, the total capacity of the open depots may be insufficient to carry the total flow. An *artificial depot* of

infinite capacity is thus added to the network and linked to all customers with high cost arcs, which ensure that no flow will go through these arcs unless there is no feasible alternative. In this way, the search is allowed to explore the infeasible domain (although it always keeps track of the best feasible solution).

In the following, we first describe in details the neighborhood structure and the global search strategy. Then, in the next subsection, we present the techniques used to reduce the neighborhood size and to evaluate exactly only a small subset of neighboring solutions at each iteration. These neighborhood reduction techniques make the tabu search procedure computationally efficient, and differentiate the approach from other tabu search heuristics previously proposed for the uncapacitated version of the problem.

3.1 Neighborhood Structure and Search Strategy

Two different neighborhoods are explored. The first neighborhood consists of moves where a single y variable is modified by opening a closed depot (ADD) or by closing an open depot (DROP). Its complexity is $O(m)$, where m is the number of depots. The second neighborhood is based on SWAP moves where two variables are modified by simultaneously closing a depot and opening another one. This neighborhood is of complexity $O(m^2)$.

For most problems, good solutions have approximately the same number of open depots. The ADD/DROP neighborhood is thus a good medium to find an appropriate number of open depots and get close to high quality solutions. However, it is not indicated for searching the space of configurations once a certain quality level has been reached, since the objective tends to vary widely when a depot is opened or closed. The SWAP neighborhood is better suited for this task, once the "right" number of open depots has been determined.

The global search strategy alternates between the two neighborhoods. For the ADD/DROP neighborhood, a tabu search is performed. In the case of the SWAP neighborhood, which is more computationally expensive than the ADD/DROP neighborhood, a pure descent is performed (i.e., the search stops at the first local minimum). Preliminary experiments have shown that performing a tabu search with SWAP moves does not bring much improvement and consumes a lot of computation time. A descent with SWAP is, however, essential to the good performance of the algorithm. It allows the method to find a good configuration, for a given number of open depots, and to "set the stage" for a new round of ADD/DROP moves.

In the description below, s is the current solution, s^* is the best solution found from the start, and s_l^* is the best solution found during each execution of the ADD/DROP tabu search. In Steps 2.2 and 2.3, both s and s^* are appropriately updated by the ADD/DROP and SWAP operators after each move. Also, s_l^* is updated after each move in Step 2.2.

1. Find an initial solution using the slope scaling procedure (presented in Section 4) and set it as the current solution s; $s^* \leftarrow s$.
2. Repeat until at least I moves (ADD/DROP and SWAP) are performed or until a CPU time limit is achieved:
   - 2.1 $s_l^* \leftarrow +\infty$.
   - 2.2 Perform ADD/DROP tabu search until $I_{A/D}$ consecutive iterations are performed without any improvement to s_l^*.
   - 2.3 Perform a SWAP descent until a local minimum is reached.

The tabu list T records the $|T|$ last depots added or dropped from the solution. This prevents the reversal of their status as long as they remain in the list. Since a SWAP move might be seen as a

combination of one ADD move and one DROP move, the depots involved in any SWAP move are also recorded in the tabu list. Thus, at every call to the ADD/DROP tabu search, the tabu list is kept in its current state and is not reinitialized. The length of the tabu tenure is a random value chosen within a user-supplied interval.

In principle, the evaluation of each move, either ADD/DROP or SWAP, involves the solution of the associated MMCF, which is computationally heavy. In the following, we describe neighborhood reduction techniques aimed at focusing only on a small subset of moves at each iteration, while retaining the effectiveness of the solution strategy.

3.2 Neighborhood Reduction

Since exact evaluations of all neighboring solutions are computationally expensive, approximation techniques are used to quickly filter out the possible moves at each iteration of the tabu search. Basically, the moves are ranked according to these approximations, and only some moves selected among the best ones are evaluated exactly, by solving an MMCF for each one. These approximations are based on estimating the costs incurred by moving flows to account for opening (ADD move) or closing (DROP move) a depot. Thus, in the case of an ADD move, we attempt to move customer-depot and depot-customer flows to the newly opened depot in order to save as much as possible on the total cost. Conversely, in the case of a DROP move, the approximation procedure moves customer-depot and depot-customer flows out of the newly closed depot, and reassigns these flows to other depots. Since a SWAP move is a combination of a DROP move and an ADD move, the same approximations are also used for the SWAP moves.

When reassigning the flows, the approximation procedure makes use of the *residual capacity* $\bar{q}_j$ at each depot j, which is defined for the current solution $\bar{x}$ (feasible for MMCF) as:

$$\bar{q}_j = q_j - \sum_{k \in K} v^k \left(\sum_{i \in C_j^*} \bar{x}_{ij}^k + \sum_{l \in D_j^-} \bar{x}_{lj}^k \right), \quad \forall j \in D. \tag{14}$$

The procedure adjusts the residual capacity of any depot j whenever some flow is reassigned to depot j, and always maintains $\bar{q}_j \geq 0$. Because the approximation procedure considers the reassignment of customer-depot and depot-customer flows independently, and does not maintain flow conservation at the depots, two residual capacities are associated with each depot j, one for the inflow and one for the outflow, which are noted $\bar{q}_j^-$ and $\bar{q}_j^+$, respectively.

Because the residual capacities are taken into account, the order in which the depots are considered for a potential flow reassignment is of prime importance. Hence, for each type of move, an "a priori" ordering of the depots adjacent to each customer $i \in C$ is used, based on the following *proximity* measures:

$$p_i^+(j) = \sum_{k \in K} c_{ij}^k, \quad \forall j \in D_i^+, \tag{15}$$

$$p_i^-(j) = \sum_{k \in K} c_{ji}^k, \quad \forall j \in D_i^-.$$

Let us assume that for each customer i, the $|D_i^+|$ and $|D_i^-|$ adjacent depots are ordered in non-increasing values of $p_i^+(j)$ and $p_i^-(j)$ (with ties broken arbitrarily). The depots adjacent to customer i are then sequenced according to this ordering, using the index sets $\{1, ..., |D_i^+|\}$ and $\{1, ..., |D_i^-|\}$. When evaluating an ADD move, the approximation procedure then attempts to move the flow from the farthest depots to the newly opened depot. Similarly, when evaluating a DROP move, the procedure attempts to move the flow from the newly closed depot to the closest open depots.

The flow reassignment also takes into account the flow transiting through the artificial depot (denoted with index 0). For example, when an ADD move is evaluated, the approximation procedure first attempts to move the flow from the artificial depot to the newly opened depot before taking care of the flow from the other depots. Conversely, for a DROP move, the flow from the newly closed depot is reassigned to the artificial depot when all other possible reassignments have already been considered.

We now describe the approximation procedure for each type of move, ADD, DROP, and SWAP. To this end, we use the following notation: $a^+ = \max\{0, a\}$; $Z_{A(j^1)}$, $Z_{D(j^0)}$ and $Z_{S(j^0,j^1)}$ are the values associated with an ADD, DROP or SWAP move, where j^0 is the depot to be closed and j^1 is the depot to be opened.

3.2.1 Approximation for the ADD move

When a depot j^1 is to be opened, new *saving* opportunities are offered to all customers adjacent to that depot. We compute these potential savings for each customer i adjacent to j^1 using the formulas:

$$u_i^+ = \sum_{k \in K} \sum_{j \in D_i^+} (c_{ij}^k - c_{ij^1}^k)^+ \overline{x}_{ij}^k, \quad \forall i \in C_{j^1}^s, \tag{16}$$

$$u_i^- = \sum_{k \in K} \sum_{j \in D_i^-} (c_{ji}^k - c_{j^1 i}^k)^+ \overline{x}_{ji}^k, \quad \forall i \in C_{j^1}^d.$$

We then attempt to reassign the customer-depot and depot-customer flows by scanning the customers in non-increasing order of their savings (with ties broken arbitrarily).

Thus, given a current solution $(\overline{x}, \overline{y})$, we can summarize the approximation procedure for an ADD move as follows:

1. $Z_{A(j^1)} = f_{j^1}$.
2. $\overline{q}_{j^1}^- = q_{j^1}$.
3. For each $i \in C_{j^1}^s$, compute the saving u_i^+.
4. Sort the customers $i \in C_{j^1}^s$ in non-increasing order of their savings u_i^+; let $\{1, .., |C_{j^1}^s|\}$ be the index set associated with this ordering.
5. For each $i = 1, ..., |C_{j^1}^s|$ do:
   5.1 First for the artificial depot ($j = 0$), then for each $j = |D_i^+|, .., 1$ such that $\overline{y}_j = 1$, do:
      5.1.1 For each $k \in K$ such that $\overline{x}_{ij}^k > 0$ do:
         5.1.1.1 If $c_{ij}^k > c_{ij^1}^k$, reassign $\widetilde{x} = \max\{\frac{\overline{q}_{j^1}^-}{v^k}, \overline{x}_{ij}^k\}$ units of flow from j to j^1: $\overline{q}_{j^1}^- = \overline{q}_{j^1}^- - v^k \widetilde{x}$; $Z_{A(j^1)} = Z_{A(j^1)} - (c_{ij}^k - c_{ij^1}^k)\widetilde{x}$.
6. $\overline{q}_{j^1}^+ = q_{j^1}$.
7. For each $i \in C_{j^1}^d$, compute the saving u_i^-.
8. Sort the customers $i \in C_{j^1}^d$ in non-increasing order of their savings u_i^-; let $\{1, .., |C_{j^1}^d|\}$ be the index set corresponding to this ordering.
9. For each $i = 1, .., |C_{j^1}^d|$ do:
   9.1 First for the artificial depot ($j = 0$), then for each $j = |D_i^-|, .., 1$ such that $\overline{y}_j = 1$, do:

9.1.1 For each $k \in K$ such that $\overline{x}^k_{ji} > 0$ do:

9.1.1.1 If $c^k_{ji} > c^k_{j^1 i}$, reassign $\widetilde{x} = \max\{\frac{\overline{q}^+_{j^1}}{v^k}, \overline{x}^k_{ji}\}$ units of flow from j to j^1: $\overline{q}^+_{j^1} = \overline{q}^+_{j^1} - v^k\widetilde{x}$; $Z_{A(j^1)} = Z_{A(j^1)} - (c^k_{ji} - c^k_{j^1 i})\widetilde{x}$.

Remarks:

- In steps 2 and 6, we initialize the residual capacities of depot j^1 only, since j^1 is the only depot which receives additional flow, as a result of the procedure. Initially, the residual capacities of j^1 are set to its full capacity, since there is no flow in-transit through j^1.
- In steps 5.1.1 and 9.1.1, the order in which we consider the commodities is of little importance, as we observed that there is typically only a few (between 1 and 4) commodities used on every arc for our test instances. If necessary, we could have scanned the commodities in non-increasing order of $v^k x^k_{ij}$ ($v^k x^k_{ji}$), but this would have increased the computational time, without improving significantly the overall performance.
- In steps 5.1.1.1 and 9.1.1.1, it is not necessary to actually reassign the flow (i.e., to compute new values for $\overline{x}$) because the only values that need to be updated are the residual capacities of j^1.

3.2.2 Approximation for the DROP move

When a depot j^0 is to be closed, every customer with some flow in-transit through depot j^0 must reassign that flow to some other open depots. To this end, the customers are handled one by one, using an order determined by *regret* measures. A large regret measure for a customer i indicates that its flow must be reassigned as soon as possible to the closest alternative open depots (as measured by their proximity), otherwise large additional costs are incurred. For each customer i adjacent to j^0, let $\overline{D}^+_i$ and $\overline{D}^-_i$ represent the sets of open depots adjacent to i, excluding j_0, and assume that these sets are ordered in non-increasing values of $p^+_i(j)$ and $p^-_i(j)$, respectively. According to this ordering, we use the index sets $\{1, ..., \overline{D}^+_i\}$ and $\{1, ..., \overline{D}^-_i\}$. The regret measures are then computed as follows:

$$r^+_i = \sum_{k \in K} \sum_{j=2}^{r^+} (c^k_{ij} - c^k_{i1})^+ \overline{x}^k_{ij^0}, \quad \forall i \in C^s_{j^0}, \tag{17}$$

$$r^-_i = \sum_{k \in K} \sum_{j=2}^{r^-} (c^k_{ji} - c^k_{1i})^+ \overline{x}^k_{j^0 i}, \quad \forall i \in C^d_{j^0}.$$

where $r^+ = \min(r_{max}, |\overline{D}^+_i|)$, $r^- = \min(r_{max}, |\overline{D}^-_i|)$ and r_{max} is a parameter usually set to a small value (i.e., if $r_{max} = 2$, only the difference in cost between the two closest depots is considered). In our experiments, r_{max} was set to 5.

The approximation procedure for the DROP move can be summarized as follows:

1. $Z_{D(j^0)} = -f_{j^0}$.
2. For each $j \in D$ such that $\overline{y}_j = 1$, except j^0, do: $\overline{q}^-_j = \overline{q}_j$.
3. For each $i \in C^s_{j^0}$, compute the regret r^+_i.
4. Sort the customers $i \in C^s_{j^0}$ in non-increasing order of their regrets r^+_i; let $\{1, .., |C^s_{j^0}|\}$ be the index set corresponding to this ordering.

5. For each $i = 1, .., |C^s_{j^0}|$ do:

   5.1 For each $k \in K$ such that $\overline{x}^k_{ij^0} > 0$ do:

      5.1.1 While $\overline{x}^k_{ij^0} > 0$, do the following, first for each $j = 1, .., |\overline{D}^+_i|$, then for the artificial depot $(j = 0)$:

         5.1.1.1 Reassign $\widetilde{x} = \max\{\frac{\overline{q}^-_j}{v^k}, \overline{x}^k_{ij^0}\}$ units of flow from j^0 to j: $\overline{q}^-_j = \overline{q}^-_j - v^k\widetilde{x}$; $Z_{D(j^0)} = Z_{D(j^0)} + (c^k_{ij^0} - c^k_{ij})\widetilde{x}$; $\overline{x}^k_{ij^0} = \overline{x}^k_{ij^0} - \widetilde{x}$; $\overline{x}^k_{ij} = \overline{x}^k_{ij} + \widetilde{x}$.

6. For each $j \in D$ such that $\overline{y}_j = 1$, except j^0, do: $\overline{q}^+_j = \overline{q}_j$.

7. For each $i \in C^d_{j^0}$, compute the regret r^-_i.

8. Sort the customers $i \in C^d_{j^0}$ in non-increasing order of their regrets r^-_i; let $\{1, .., |C^d_{j^0}|\}$ be the index set corresponding to this ordering.

9. For each $i = 1, .., |C^d_{j^0}|$ do:

   9.1 For each $k \in K$ such that $\overline{x}^k_{j^0_i} > 0$ do:

      9.1.1 While $\overline{x}^k_{j^0_i} > 0$, do the following, first for each $j = 1, .., |\overline{D}^-_i|$, then for the artificial depot $(j = 0)$:

         9.1.1.1 Reassign $\widetilde{x} = \max\{\frac{\overline{q}^+_j}{v^k}, \overline{x}^k_{j^0_i}\}$ units of flow from j^0 to j: $\overline{q}^+_j = \overline{q}^+_j - v^k\widetilde{x}$; $Z_{D(j^0)} = Z_{D(j^0)} + (c^k_{j^0_i} - c^k_{ji})\widetilde{x}$; $\overline{x}^k_{j^0_i} = \overline{x}^k_{j^0_i} - \widetilde{x}$; $\overline{x}^k_{ji} = \overline{x}^k_{ji} + \widetilde{x}$.

Remarks:

- In steps 2 and 6, we initialize the residual capacities of all open depots except j^0, since all might receive additional flow.
- In steps 5.1 and 9.1, the order in which the commodities are considered is of little importance for the same reasons mentioned in the case of an ADD move.
- In steps 5.1.1.1 and 9.1.1.1, it is now necessary to compute new values for $\overline{x}$. First, we note that the value of the flow through j^0 must be updated, as it is used to stop the reassignment. Second, it is true that the flow through any open depot j is never used by the procedure, but it will be used by the approximation procedure for the SWAP move (see below). This approximation assumes that a SWAP move is made of a DROP, followed by an ADD. Therefore, the information gathered by the DROP approximation (i.e., the flows through the open depots) is transferred to the ADD approximation procedure, which makes use of it.

3.2.3 Approximation for the SWAP move

In the case of a SWAP move, we first rank the depots j^0 to be closed, using the approximation described for the DROP move. Then, for each closed depot j^0, we rank the depots j^1 to be opened, using the approximation described for the ADD move. Thus, all possible SWAP moves are ranked according to the value $Z_{S(j^0,j^1)} = Z_{D(j^0)} + Z_{A(j^1)}$.

3.2.4 Exact Evaluation

In the case of the ADD/DROP neighborhood, we evaluate exactly n_{ADD} moves produced by ADD and n_{DROP} moves produced by DROP. In the case of the SWAP neighborhood, n_{SWAP} moves are evaluated

exactly. However, we do not necessarily select the moves associated with the best approximation values for this purpose. We rather use a probabilistic perturbation scheme, biased towards the best approximations. It proves to be useful by allowing the algorithm to perform, from time to time, a move that is not necessarily among the best ones.

Assuming r different moves, either of the ADD, DROP or SWAP types, the move with the best approximation value (rank 1) is associated with some Z_{max} value, while the worst one (rank r) is associated with some Z_{min} value. The values of the other moves are then equally spaced between Z_{min} and Z_{max}. The value Z_i for the move of rank i is computed as:

$$Z_i = Z_{max} - (Z_{max} - Z_{min}) \frac{i-1}{r-1}, \quad 1 \leq i \leq r.$$

The probability p_i of selecting the move of rank i is then:

$$p_i = \frac{Z_i}{\sum_{j=1}^{r} Z_j}, \quad 1 \leq i \leq r.$$

Assuming that $Z_{min} + Z_{max} = 2$, the selection bias in favor of the best approximations can be increased by setting the Z_{max} value closer to 2, or reduced by setting its value closer to 1 [14]. In our computational results, we used $Z_{min} = 0.5$ and $Z_{max} = 1.5$.

A total of $n_{ADD} + n_{DROP}$ or n_{SWAP} selection trials (depending on the neighborhood, ADD/DROP or SWAP) are performed, based on this probability distribution. The selected moves are then evaluated exactly, by solving an MMCF for each of them. Among all moves evaluated exactly, the best one is chosen at the end.

4 Slope Scaling

A slope scaling procedure is used to produce an initial solution for the tabu search (see [10, 11, 12] for recent successful applications of this type of heuristic for solving non-convex piecewise linear network flow problems). This is an iterative procedure, where successive MMCFs are solved, based on modified linear costs. In our context, all depots are implicitly open in the formulation of the MMCF (i.e., $\bar{y}_j = 1$, $\forall j \in D$), but the linear costs, $\bar{c}$, are modified to reflect the contribution of the fixed costs.

More specifically, given a solution $\widetilde{x}$ to some MMCF formulation, and assuming that $\widetilde{y}$ is computed according to (13), the linear transportation costs are modified so that

$$\sum_{k \in K} \left(\sum_{(i,j) \in A_{CD}} \bar{c}_{ij}^k \widetilde{x}_{ij}^k + \sum_{(j,i) \in A_{DC}} \bar{c}_{ji}^k \widetilde{x}_{ji}^k + \sum_{(l,j) \in A_{DD}} \bar{c}_{lj}^k \widetilde{x}_{lj}^k \right) = Z(\widetilde{x}, \widetilde{y}), \tag{18}$$

where $Z(\widetilde{x}, \widetilde{y})$ is the total cost of the feasible solution $(\widetilde{x}, \widetilde{y})$ given by equation (12). In other words, the goal of the approach is to solve an MMCF with modified costs $\bar{c}$ so that, if the solution remains the same, the costs $\bar{c}$ reflect exactly the total cost, both linear and fixed, incurred by this solution. To compute the modified costs, we use the total volume in-transit through each depot j, defined for any feasible flow $\widetilde{x}$, as:

$$\widetilde{X}_j = \sum_{k \in K} v^k \left(\sum_{i \in C_j^-} \widetilde{x}_{ij}^k + \sum_{l \in D_j^-} \widetilde{x}_{lj}^k \right). \tag{19}$$

The modification to the linear costs proceeds as follows. We denote the modified cost associated with arc (i,j) and commodity k at iteration $t \geq 0$ as $\bar{c}_{ij}^{k(t)}$. Similarly, we denote by $\widetilde{X}_j^t$ the total volume

in-transit through depot j at iteration $t \geq 0$. Initially, at iteration 0, we set the modified costs as follows:

$$\bar{c}_{ij}^{k(0)} = c_{ij}^k + v^k \alpha_j \frac{f_j}{q_j}, \quad \forall (i,j) \in A_{CD}, k \in K, \tag{20}$$

$$\bar{c}_{ji}^{k(0)} = c_{ji}^k + v^k (1-\alpha_j) \frac{f_j}{q_j}, \quad \forall (j,i) \in A_{DC}, k \in K, \tag{21}$$

$$\bar{c}_{lj}^{k(0)} = c_{lj}^k + v^k \left(\alpha_j \frac{f_j}{q_j} + (1-\alpha_l) \frac{f_l}{q_l} \right), \quad \forall (l,j) \in A_{DD}, k \in K, \tag{22}$$

where $\alpha_j = \frac{\xi_j}{\xi_j + \Delta_j}$ with $\xi_j = \sum_{k \in K} v^k \sum_{i \in C_j^s} s_i^k$ and $\Delta_j = \sum_{k \in K} v^k \sum_{i \in C_j^d} d_i^k$, for each $j \in D$. Here, ξ_j (Δ_j) is the maximum volume that might transit through depot j from all supply (demand) customers adjacent to it. Parameter α_j ($1 - \alpha_j$) thus approximates the fraction of the total volume in-transit at depot j which can be imputed to supply (demand) customers, and is used to calibrate the cost on the corresponding arcs.

Note that these initial modified costs are derived from the LP relaxation of the MCLB formulation with the redundant constraints (6) and (7) removed. It is easy to show that any optimal solution to this LP must satisfy, for each depot j,

$$y_j = \frac{1}{q_j} \sum_{k \in K} v^k \left(\sum_{i \in C_j^s} x_{ij}^k + \sum_{l \in D_j^+} x_{lj}^k \right) \tag{23}$$

$$= \frac{1}{q_j} \sum_{k \in K} v^k \left(\alpha_j \left(\sum_{i \in C_j^s} x_{ij}^k + \sum_{l \in D_j^+} x_{lj}^k \right) + (1-\alpha_j) \left(\sum_{i \in C_j^d} x_{ji}^k + \sum_{l \in D_j^-} x_{jl}^k \right) \right). \tag{24}$$

By replacing the y variables in the objective with this last expression, we obtain the MMCF with costs given by formulas (20)-(22).

Based on the optimal solution to the MMCF with modified costs obtained at the previous iteration, the linear costs are updated as follows at every iteration $t > 0$. First, we consider the arcs for which *all* incident depots are used for in-transit flows (i.e., $\widetilde{X}_j^{t-1} > 0$ for all incident depots j):

$$\bar{c}_{ij}^{k(t)} = c_{ij}^k + v^k \alpha_j \frac{f_j}{\widetilde{X}_j^{t-1}}, \quad \forall (i,j) \in A_{CD}, k \in K, \tag{25}$$

$$\bar{c}_{ji}^{k(t)} = c_{ji}^k + v^k (1-\alpha_j) \frac{f_j}{\widetilde{X}_j^{t-1}}, \quad \forall (j,i) \in A_{DC}, k \in K, \tag{26}$$

$$\bar{c}_{lj}^{k(t)} = c_{lj}^k + v^k \left(\alpha_j \frac{f_j}{\widetilde{X}_j^{t-1}} + (1-\alpha_l) \frac{f_l}{\widetilde{X}_l^{t-1}} \right), \quad \forall (l,j) \in A_{DD}, k \in K. \tag{27}$$

It is easy to verify that this cost update satisfies (18). For arcs with *at least one* unused incident depot, we cannot apply formulas (25)-(27), since $\widetilde{X}_j^{t-1} = 0$ for at least one incident depot j. In this case, the costs of the corresponding arcs are updated as follows at iteration $t > 0$:

$$\bar{c}_{ij}^{k(t)} = \beta \max\{\widetilde{c}^{(0)}, \max_{0<\tau<t} \{\bar{c}_{ij}^{k(\tau)} | \widetilde{X}_j^\tau > 0\}\}, \quad \forall (i,j) \in A_{CD}, k \in K, \tag{28}$$

$$\bar{c}_{ji}^{k(t)} = \beta \max\{\widetilde{c}^{(0)}, \max_{0<\tau<t} \{\bar{c}_{ji}^{k(\tau)} | \widetilde{X}_j^\tau > 0\}\}, \quad \forall (j,i) \in A_{DC}, k \in K, \tag{29}$$

$$\bar{c}_{lj}^{k(t)} = \beta \max\{\widetilde{c}^{(0)}, \max_{0<\tau<t} \{\bar{c}_{lj}^{k(\tau)} | \widetilde{X}_j^\tau, \widetilde{X}_l^\tau > 0\}\}, \quad \forall (l,j) \in A_{DD}, k \in K, \tag{30}$$

where $\widetilde{c}^{(0)} = \max_{(i,j) \in A, k \in K} \bar{c}_{ij}^{k(0)}$, and $\beta > 0$ is a parameter. When $\beta = 1$, this formula sets the arc cost either to the largest cost at iteration 0, or to the largest arc cost value from the previous iterations

which led to the use of all incident depots for in-transit flows (a similar rule is used in [10]). Setting β to a large value virtually closes the corresponding arc, while this decision can possibly be reverted in the following iterations when β is relatively small. In our implementation, β was set to a large value, as we were mostly interested to quickly identify a "good" feasible solution. Typically, setting β to a small value consumes significantly more time. Preliminary experiments revealed that the value $\beta = 1000$ produced the best results when the slope scaling procedure is used during the initialization phase of the tabu search.

The slope scaling procedure is stopped when there are no modifications in the costs from one iteration to the next. At each iteration, a feasible solution $\widetilde{x}$ is obtained, and an upper bound is computed according to equation (12). The best upper bound found so far is kept in variable Z^*. If we denote by $\widetilde{x}$ the solution provided by the slope scaling procedure, the corresponding depot configuration, $\widetilde{y}$, is then computed with (13). Recall that this solution and hence the resulting depot configuration are derived from a linear approximation of the fixed and transportation costs. The resulting flow structure is therefore optimal with respect to this approximation but generally not with respect to the original transportation costs. Thus, before starting the tabu search, one needs to optimize the flow structure given the depot configuration $\widetilde{y}$. To do so, we solve another MMCF using the original costs (i.e. $\bar{c} = c$) and the depot configuration $\widetilde{y}$ (i.e. $\bar{y} = \widetilde{y}$), thus obtaining a new feasible solution $\widetilde{x}$. The initial solution provided to the tabu search is then derived from (13), with its value $Z(\widetilde{x}, \widetilde{y})$ provided by (12). If $Z(\widetilde{x}, \widetilde{y})$ improves upon Z^*, which is often the case, it replaces it.

5 Computational Results

In this section, we report computational results on problem instances obtained with the generator described in the Appendix. The solutions produced by the heuristic are compared with optimal solutions obtained with the MIP solver CPLEX, version 6.6 [9]. All tests were run on 400MHz UltraSparc2 processors.

In the following, the problem instances are first described. Then, subsection 5.2 provides some insights about the quality of the approximation procedures for the ADD and DROP moves (see Section 3.2). Then, we analyze the performance of the heuristic and compare it to CPLEX.

5.1 Problem Instances

The heuristic was evaluated on different types of problems produced with our generator. A problem type is defined through the following parameter values (for detailed explanations about parameters n_h, n_v, s_f, $\bar{d}$ and γ, the reader is referred to the Appendix):

- *Number of customers* n: 500.
- *Number of depots* m: 200.
- *Number of commodities* p: 20.
- *Number of customer zones* $n_h \times n_v$: two types of instances are generated, using either a 3×2 grid (dense) or a 4×3 grid (sparse).
- *Fixed cost multiplier* s_f: two values are used to multiply the original fixed costs, 1000 and 5000.
- *Mean demand value for each commodity* $\bar{d}$: 100.
- *Capacity tightness parameter* γ: all instances are capacitated, using $\gamma = 0.3$.

Thus, four different types of problems are considered, depending on the number of customer zones and the fixed cost multipliers. Preliminary experiments have shown that the performance of the heuristic is particularly sensitive to these characteristics. By reporting results obtained with different values for these critical parameters, some interesting trends can be revealed. For each type of problems, five different instances were generated.

5.2 Quality of the Approximation Procedures

We have examined how well the approximation procedures developed for the ADD and DROP moves (see Section 3.2) are consistent with the exact evaluations obtained with CPLEX. In a typical run, we observed that the best ADD move was found in the 2%-3% top ranked ADD approximations, while the best DROP move was found in the 15%-20% top ranked DROP approximations. Thus, the ADD approximation is much more accurate than the DROP approximation. This is not a surprise, given the difficulty of heuristically reassigning the flow to the remaining open depots when a DROP move is applied. This observation is used in the next section for selecting the parameter values of the tabu search.

5.3 Performance Analysis and Comparison with CPLEX

Computational results are reported on the four types of problems introduced in Section 5.1. The parameter settings for the tabu search are as follows:

Number of iterations: I, $I_{A/D}$. Preliminary experiments have shown that I=300 iterations provide a good trade-off between solution quality and computation time. In fact, most of the optimization occurs during those first 300 iterations, as it will be illustrated with results obtained when the tabu search runs much longer. Parameter $I_{A/D}$ was set to 10. We observed that smaller values are costly, in the sense that they often prevent the search to identify high quality solutions in the current region of the search space. Conversely, larger values tend to be unproductive.

Neighborhood size: n_{ADD}, n_{DROP}, n_{SWAP}. The number of exact evaluations for ADD should be small, given the quality of the approximation (see Section 5.2). Thus, we used the value $n_{ADD} = 3$. We also used $n_{DROP} = 40$, i.e., the number of exact evaluations for DROP is an order of magnitude larger than in the case of ADD. This is supported by two facts: (1) the DROP approximation is less reliable than the ADD approximation and (2) every opportunity to close a depot must be exploited to generate a good solution. This situation is similar to some vehicle routing applications, where saving a vehicle is of paramount importance due to important acquisition costs. Finally, the number of exact evaluation of SWAP moves must remain small, due to its computational requirements: we chose $n_{SWAP} = 9$.

Tabu tenure. The tabu tenure for the ADD and DROP moves is randomly selected in the interval [4,7]. Note that the types of moves considered here induce a significant impact on solution structures. This justifies the use of such relatively small tabu tenure values.

Tables 1 and 2 report optimal solutions obtained with CPLEX (with the default parameter values) on each problem instance *a_b_c*, where *a* corresponds to the $n_h \times n_v$ customer zones, *b* to the scaling factor s_f for the fixed cost, and *c* to the instance number (1 to 5). In these tables, the solution value and CPU time associated with the LP relaxation, first integer solution (FI) and optimum (OPT) are shown. The LP relaxation is solved with the combined network/dual simplex method implemented in CPLEX, using the so-called steepest-edge pricing rule (see the CPLEX documentation [9] for more details). Note that on instances *3x2_5000_1* and *3x2_5000_2*, CPLEX was stopped after 10,000,000 seconds, without a proof of optimality (in fact, the solution obtained in the first case is not optimal, as a better solution was found with the tabu search heuristic). A large variance in CPU times is observed from one instance to another, which is typical of exact methods. Clearly, CPLEX has a harder time when the fixed costs are high and when the number of zones is small.

Problem	Linear relaxation		First integer (FI)		Optimum (OPT)	
	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)
3x2_1000_1	65 252	40 724	66 102	64 149	65 606	1 077 564
3x2_1000_2	87 220	24 893	87 727	41 145	87 711	262 574
3x2_1000_3	79 368	37 626	79 571	50 867	79 496	171 790
3x2_1000_4	43 594	59 191	44 053	99 926	43 686	579 249
3x2_1000_5	52 487	49 084	53 302	85 605	52 718	3 450 811
3x2_5000_1	183 473	70 928	187 641	84 639	*186 527	10 000 000
3x2_5000_2	210 472	63 162	213 333	74 079	*211 299	10 000 000
3x2_5000_3	202 632	61 957	206 648	78 139	203 306	680 446
3x2_5000_4	151 940	110 385	154 909	125 326	152 714	2 431 508
3x2_5000_5	171 925	124 897	179 928	152 320	172 557	1 191 996

Table 1: CPLEX runs on instances with 3x2 customer zones

Problem	Linear relaxation		First integer (FI)		Optimum (OPT)	
	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)
4x3_1000_1	101 359	27 813	101 776	47 505	101 621	626 412
4x3_1000_2	143 632	4 709	144 344	12 486	144 004	133 238
4x3_1000_3	172 158	3 651	174 053	6 599	173 399	120 784
4x3_1000_4	76 212	31 028	76 741	39 609	76 544	435 680
4x3_1000_5	156 859	3 953	157 674	12 524	157 399	205 873
4x3_5000_1	210 370	23 037	212 189	28 203	211 299	186 004
4x3_5000_2	274 092	48 805	276 263	77 436	275 036	1 739 470
4x3_5000_3	298 729	24 221	302 943	30 718	299 775	173 359
4x3_5000_4	180 721	40 076	184 181	52 201	181 911	882 657
4x3_5000_5	271 451	18 624	275 785	24 651	272 483	236 901

Table 2: CPLEX runs on instances with 4x3 customer zones

Problem	CPLEX (FI)		SS		TS			
	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU* (sec.)	CPU (sec.)	OPT gap
3x2_1000_1	66 102	64 149	66 204	173	66 006	9 966	17 477	0.61%
					65 716	45 744	64 285	0.16%
3x2_1000_2	87 727	41 145	87 727	184	87 720	2 324	18 669	0.01%
					87 711	*27 538*	*41 219*	*0.00%*
3x2_1000_3	79 571	50 867	79 655	182	79 554	23 776	28 720	0.07%
					79 554	23 776	50 964	0.07%
3x2_1000_4	44 053	99 926	44 075	163	43 966	5 849	19 950	0.64%
					43 946	30 923	99 940	0.59%
3x2_1000_5	53 302	85 605	53 728	205	53 447	18 612	25 714	1.38%
					53 117	40 546	85 753	0.75%
Average	66 151	68 338	66 277	181	66 138	12 105	18 515	0.54%
					66 008	33 705	68 432	0.31%
3x2_5000_1	187 641	84 639	188 070	185	186 807	19 509	28 027	*0.15%
					186 807	19 509	84 733	*0.15%
3x2_5000_2	213 333	74 079	213 333	210	213 333	210	16 727	*0.96%
					211 939	52 651	74 226	*0.30%
3x2_5000_3	206 648	78 139	205 317	147	205 317	147	16 126	0.99%
					204 252	38 092	78 190	0.46%
3x2_5000_4	154 909	125 326	154 909	270	*152 714*	*14 934*	*20 742*	*0.00%*
					152 714	*14 934*	*125 328*	*0.00%*
3x2_5000_5	179 928	152 320	178 823	223	*172 557*	*366*	*26 534*	*0.00%*
					172 557	*366*	*152 377*	*0.00%*
Average	188 491	102 901	188 090	187	186 145	7 033	21 667	0.42%
					185 653	25 110	102 970	0.18%

Table 3: Tabu runs on instances with 3x2 customer zones

Problem	CPLEX (FI)		SS		TS			
	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU (sec.)	solution ($\times 10^3$)	CPU* (sec.)	CPU (sec.)	OPT gap
4x3_1000_1	101 776	47 505	104 118	82	102 081	5 319	8 901	0.45%
					102 081	5 319	47 513	0.45%
4x3_1000_2	144 344	12 486	144 209	85	144 209	85	10 889	0.14%
					144 209	85	12 503	0.14%
4x3_1000_3	174 053	6 599	174 099	80	174 099	80	7 702	0.40%
					174 099	80	6 611	0.40%
4x3_1000_4	76 741	39 609	77 011	81	76 845	6 956	7 744	0.39%
					76 832	31 460	39 649	0.37%
4x3_1000_5	157 674	12 524	159 142	76	157 853	4 719	5 696	0.28%
					157 853	4 719	12 559	0.28%
Average	130 917	23 745	131 715	80	131 017	3 431	8 186	0.33%
					131 014	8 332	23 767	0.33%
4x3_5000_1	212 189	28 203	218 332	81	212 394	9 304	11 926	0.51%
					212 394	9 304	28 204	0.51%
4x3_5000_2	276 263	77 436	275 286	109	275 286	109	9 793	0.09%
					275 286	109	77 459	0.09%
4x3_5000_3	302 943	30 718	*299 775*	*105*	*299 775*	*105*	*11 249*	*0.00%*
					299 775	*105*	*30 776*	*0.00%*
4x3_5000_4	184 181	52 201	184 596	67	183 919	2 458	6 604	1.10%
					181 928	52 008	52 207	0.01%
4x3_5000_5	275 785	24 651	279 195	74	274 951	5 013	6 304	0.90%
					274 802	6 304	24 667	0.85%
Average	250 272	42 642	251 436	87	249 265	3 397	9 175	0.52%
					248 837	13 566	42 662	0.29%

Table 4: Tabu runs on instances with 4x3 customer zones

In Tables 3 and 4, the results produced by our tabu search with slope scaling are reported. For each problem instance, we show the solution value and CPU time for the first integer solution found by CPLEX (*CPLEX (FI)*, as taken from Tables 1 and 2); the initial solution produced by the slope scaling procedure (*SS*); and the solution value, CPU time to best solution (*CPU**), total CPU time and optimality gap for the tabu search (*TS*) after $I = 300$ iterations. For each type of problem, averages taken over the five instances are also reported. Since *CPLEX (FI)* might be seen as a heuristic competitor of *TS*, the values obtained when the tabu search is run as long as *CPLEX (FI)* are shown on the second line for each problem instance. Note that a star is found besides the optimality gaps of instances *3x2_5000_1* and *3x2_5000_2* to indicate that this is the gap with respect to the best known solution (given that the optimum is not known). Optimal solutions found by the heuristic procedures are highlighted in *italic*.

We observe that the dense 3x2 instances are much harder to solve than the sparse 4x3 instances, as revealed by the CPU times. In particular, the time consumed in solving the MMCFs within the tabu search is significantly larger for dense instances (for the same number of iterations, 300, the 3x2 instances require more computational effort than the 4x3 instances). We note however that the heuristic is equally effective on both types of instances. In particular, the slope scaling procedure displays an average optimality gap of 1.18% on the 3x2 instances, compared to 1.32% on the 4x3 instances. Similarly, the tabu search identifies solutions of similar quality after 300 iterations, irrespective of the number of customer zones: the average optimality gap is 0.48% for the 3x2 instances, and 0.43% for the 4x3 instances.

To confirm these tendencies with regard to the number of customer zones, we have performed experiments on a limited number of 6x4 and 8x6 instances (two for each class, one with $s_f = 1000$ and one with $s_f = 5000$). The results support our previous observations: when comparing the results with those of Tables 3 and 4, we note that the CPU times decrease significantly, especially for CPLEX, and that the slope scaling procedure is equally effective: it identifies solutions that are, on average, within 2% of the optimum. Since CPLEX runs significantly faster on sparser instances, one might ask whether the first integer solutions it finds outperform the solutions produced by the tabu search. On the contrary, the tabu search remains competitive with *CPLEX (FI)* on these four instances after 300 iterations: on average, *TS* shows an optimality gap of 0.64%, with an average CPU time of 4 492 sec., compared to an optimality gap of 0.89%, with an average CPU time of 7 333 sec., for *CPLEX (FI)*. Even if these four instances are significantly easier to solve than the ones shown in Tables 3 and 4, we note that computing an optimal solution is still a very hard task, since CPLEX spent, on average, 1 251 706 seconds before it identified an optimal solution for each of these four instances.

With regard to the fixed cost multipliers, we observe from Tables 3 and 4 that the instances with $s_f = 5000$ are significantly more difficult to solve than the ones with $s_f = 1000$. This can be seen from the CPU times, but also from the quality of the solutions obtained. For example, the slope scaling procedure shows an average optimality gap of 0.86% when $s_f = 1000$, compared to 1.52% when $s_f = 5000$. In spite of these differences in the quality of the initial solution among the two types of instances, the tabu search identifies solutions of about the same quality for both $s_f = 1000$ and $s_f = 5000$: when $s_f = 1000$, the average optimality gap is 0.44%, compared to 0.47% when $s_f = 5000$.

After 300 terations, when compared to *CPLEX (FI)*, *TS* provides solutions of similar quality in much less CPU time (in particular, when s_f=5000). By running the tabu search as long as *CPLEX (FI)*, which often represents a substantial increase in CPU time, the tabu search can significanlty improve upon *CPLEX (FI)*, especially for 3x2 instances. Finally, we note that the tabu search generally identifies its best solution relatively late during its execution, as revealed by CPU*, which indicates that the heuristic is effective in exploring the solution space.

6 Conclusion

We have presented a tabu search heuristic, combined with a slope scaling procedure, for solving the multicommodity capacitated location problem with balancing requirements (MCLB). The tabu search is especially efficient when the customers are distributed over a small number of geographic zones, and when the fixed costs for opening and operating the depots are large. In these cases, it provides a means to generate solutions of high quality in relatively short computation times.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is hereby gratefully acknowledged. Thanks also to Serge Bisaillon for running the computational experiments.

References

[1] Crainic T.G., Dejax P.J., Delorme L., "Models for Multimode Multicommodity Location Problems with Interdepot Balancing Requirements", *Annals of Operations Research 18*, 279–302, 1989.

[2] Crainic T.G., Delorme L., "Dual-Ascent Procedures for Multicommodity Location-Allocation Problems with Balancing Requirements", *Transportation Science 27*, 90–101, 1993.

[3] Crainic T.G., Delorme L., Dejax P.J., "A Branch-and-Bound Method for Multicommodity Location with Balancing Requirements", *European Journal of Operational Research 65*, 368–382, 1993.

[4] Crainic T.G., Gendreau M., Soriano P., Toulouse M., "A Tabu Search Procedure for Multicommodity Location/Allocation with Balancing Requirements", *Annals of Operations Research 41*, 359–383, 1993.

[5] Gendron B., Crainic T.G., "A Branch-and-Bound Algorithm for Depot Location and Container Fleet Management", *Location Science 3*, 39–53, 1995.

[6] Gendron B., Crainic T.G., "A Parallel Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements", *Computers & Operations Research 24*, 829–847, 1997.

[7] Gendron G., Potvin J.-Y., Soriano P., "Tabu Search with Exact Neighbor Evaluation for Multicommodity Location with Balancing Requirements", *INFOR 37*, 255–270, 1999.

[8] Glover F., Laguna M., *Tabu Search*, Kluwer, 1997.

[9] ILOG, *ILOG CPLEX 6.6*, 1999.

[10] Kim D., Pardalos P.M., "A Solution Approach to the Fixed Charge Network Flow Problem using a Dynamic Slope Scaling Procedure", *Operations Research Letters 24*, 195–203, 1999.

[11] Kim D., Pardalos P.M., "Dynamic Slope Scaling and Trust Interval Techniques for Solving Concave Piecewise Linear Network Flow Problems", *Networks 35*, 216-222, 2000.

[12] Kim D., Pardalos P.M., "A Dynamic Domain Contraction Algorithm for Nonconvex Piecewise Linear Network Flow Problems", *Journal of Global Optimization 17*, 225-234, 2000.

[13] Sridharan R., "The Capacitated Plant Location Problem", *European Journal of Operational Research 87*, 203–213, 1995.

[14] Whitley D., "The GENITOR Algorithm: Why Rank-Based Allocation of Reproductive Trials is Best", in *Proceedings of the Third Int. Conf. on Genetic Algorithms*, J.D. Schaffer ed., Morgan Kaufmann: San Mateo, CA, 116–121, 1989.

Appendix: Problem Generator

In the following, the problem generator is described. As it contains a large number of parameters, the real values for generating our test problems are often used in place of the underlying parameters.

A.1 Network

The network lies in a rectangle divided into customer zones and depot zones. Both types of zones correspond to squares of the same dimensions in the Euclidean plane. The number of customer zones determines the dimensions of the rectangular area, while each depot zone is centered at the intersection of four customer zones (see Figure 1 for an illustration).

A.1.1 Customer Nodes

The rectangular area of interest is first divided into a certain number of customer zones, where each zone corresponds to a 100x100 square in the Euclidean plane. Given n_h zones along the horizontal axis and n_v zones along the vertical axis, the customers are then distributed among the $nc = n_h n_v$ available zones. An example is shown in Figure 1 for $nc = 6$ zones with $n_h = 3$, $n_v = 2$. We also specify a number n_{hd} of high-density, n_{md} of medium-density, and n_{ld} of low-density zones, such that $n_{hd} + n_{md} + n_{ld} = nc$. These zones are randomly chosen among the available zones. With each type of zone, we associate weights $wc_{hd} > wc_{md} > wc_{ld}$. Given n customers in set C, we then determine the number of customers assigned to zone i as:

$$n_i = \lceil \frac{wc_i}{\sum_{j=1}^{nc} wc_j} n \rceil, \; i = 1, ..., nc,$$

where wc_i is either wc_{hd}, wc_{md} or wc_{ld}, depending on the zone type. The location of a customer within a given zone is chosen randomly according to a uniform distribution.

A.1.2 Depot Nodes

Depot zones are shown with dashed lines in Figure 1. Basically, there are $n_h - 1$ zones along the horizontal axis and $n_v - 1$ zones along the vertical axis for a total of $nd = (n_h - 1)(n_v - 1)$ zones. Let us assume that any given zone j for the depots has a non empty intersection with a set K_j of customer zones. Then, given m depots, the number of depots assigned to zone j is:

$$m_j = \lceil \frac{wd_j}{\sum_{k=1}^{nd} wd_k} m \rceil, \; j = 1, ..., nd,$$

where

$$wd_j = \sum_{k \in K_j} wc_k, \; j = 1, ..., m.$$

The location of a depot within a given zone is chosen randomly according to a uniform distribution.

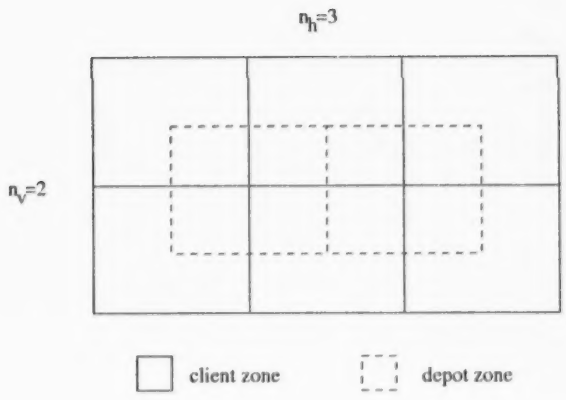

Figure 1: Customer and depot zones

A.1.3 Arcs

Having located the customer and depot nodes, the connectivity of the network is specified as follows.

Customer-Depot and Depot-Customer Arcs

Assume that customer i is located in customer zone Zc_i and depot j is located in depot zone Zd_j. Then, arcs (i, j) and (j, i) are added to the network if and only if $Zc_i \bigcap Zd_j \neq \emptyset$.

Depot-Depot Arcs

We first construct a shortest spanning tree on the complete undirected graph obtained by considering only the depot nodes, with edges weighted by the inter-depot Euclidean distances. This tree represents the backbone of the interdepot connections. Then, additional edges are added by connecting a certain percentage of pairs of leaves of this spanning tree found in adjacent depot zones. In our experiments, this percentage is set to 10%. Each edge $\{i, j\}$ obtained through this process is replaced by the directed arcs (i, j) and (j, i) in the real network.

A.2 Linear and Fixed Costs

The fixed cost to open and operate a depot is set to a basic value randomly chosen in an interval according to a uniform distribution. In our experiments, we used the interval [1000, 2000]. These values introduce some variability among the depots. They are then multiplied by a scaling factor s_f to put more or less emphasis on the fixed costs versus the linear costs.

The linear cost for each arc-commodity pair is based on the Euclidean distance multiplied by a commodity-dependent factor which, for each commodity, is an integer randomly chosen between 1 and 10 (for a given commodity, the same number is used for all arcs). For customer-depot or depot-customer arcs, the linear cost is actually the Euclidean distance between the two nodes multiplied by the commodity-dependent factor. For depot-depot arcs, the linear cost is the Euclidean distance (i.e.

two-digit precision real values) multiplied by the commodity-dependent factor and by 0.6, to take into account the economies of scale obtained through consolidation.

We also generate linear costs for the depot-depot arcs which are missing in the network. To this end, we compute the shortest paths for all pairs of depots and connect every pair of depots with a pair of arcs weighted by the shortest path value. The linear costs on these arcs are simply the shortest path value multiplied by the commodity-dependent factor. Thus, the interdepot connections form a complete directed graph. This is important, since even if some commodities are not stored at a particular depot, they might pass through them in order to save on the overall transportation costs (this situation arises in actual applications where depots correspond to rail terminals, and interdepot connections are railyards: commodities might use the railyards without stopping at the terminals).

A.3 Demand and Supply

The different customer zones used to generate the customers are also defined as being either a supply generating, demand generating or balanced zone. These three types are distributed among the high, medium and low-density zones, at their pro-rata. A mean demand value, $\bar{d}$, supplied as a parameter and which is the same for all commodities in set K, is then used to assign a demand d_i^k and a supply s_i^k to each customer i for each commodity k as follows.

When a demand is assigned to the customer, the amount is randomly chosen within the following admissible intervals, depending on the zone type:

a. demand zone $[\bar{d}+10\%, \bar{d}+30\%]$
b. supply zone $[\bar{d}-30\%, \bar{d}-10\%]$
c. balanced zone $[\bar{d}-10\%, \bar{d}+10\%]$

Conversely, when a supply is assigned to the customer, the amount is randomly chosen in the following admissible intervals, depending on the zone type:

a. demand zone $[\bar{d}-30\%, \bar{d}-10\%]$
b. supply zone $[\bar{d}+10\%, \bar{d}+30\%]$
c. balanced zone $[\bar{d}-10\%, \bar{d}+10\%]$

In a second step, a target value for the total demand (supply) of each commodity to be distributed is first defined as:

$$t = 0.8\bar{d}n.$$

Note that the fraction 0.8 is aimed at reducing the total demand (supply) to be distributed. This introduces more variability in the assignment process, by depriving a number of customers of any demand or supply for one or more commodities during the assignment procedure (see below).

After the first step, the total amount distributed for each commodity k, $\bar{d}^k$ for the demand and $\bar{s}^k$ for the supply, is typically under the target value t. The remaining demand (supply) for each commodity, $t-d^k$ $(t-s^k)$, must then be distributed among the customers. To this end, we repeat the following until the value t is reached: one customer is randomly chosen and a demand (supply) for a randomly chosen commodity is assigned to it. If some amount has already been assigned to the customer during the first step, a random fraction of the remainder, with regard to the upper bound of its admissible interval, is assigned to it. Otherwise, the amount is randomly selected within the admissible interval, as in the first step. When assigning this demand (supply), we also verify that the total demand (supply) already assigned never exceeds the target t.

A.4 Capacity

The volume of one unit of commodity $k \in K$ is an integer randomly chosen between 1 and 20. Assuming that $V = \sum_{k \in K} v^k \sum_{i \in C^+} s_i^k$ represents the maximum volume that might transit through all depots, we generate the capacities in such a way that $\sum_{j \in D} q_j \approx V/\gamma$, where $\gamma \in (0,1)$ is a user-supplied parameter. When $\gamma \to 1$ the problem is more constrained and conversely. Values between 0.2 and 0.4 are indicated: higher values lead to infeasible instances, while lower values lead to virtually uncapacitated problems.

More precisely, the capacities are generated as follows:

1. Compute a target capacity $q_\gamma = \lceil V/(\gamma \times m) \rceil$, where m is the number of depots.
2. Randomly group all depots into $\lfloor m/2 \rfloor$ pairs, except one if m is odd. In this case we set the capacity of the lone depot to q_γ.
3. For each pair of depots (j_1, j_2), compute $q_{j_1} = (1+\lambda) \times q_\gamma$ and $q_{j_2} = (1-\lambda) \times q_\gamma$, with λ randomly chosen in an interval $I_\lambda \subseteq [0,1)$ (we used $I_\lambda = [0.1, 0.5]$ for all instances).